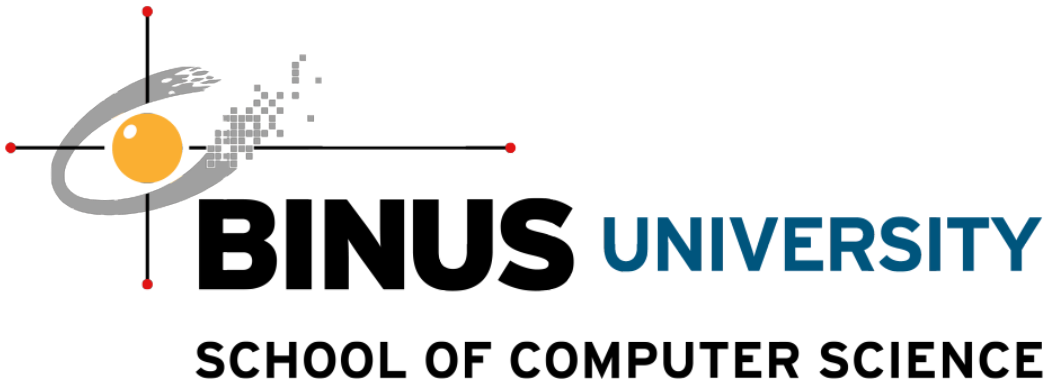
Perbandingan Kinerja Model Random Forest dan Naive Bayes dalam Prediksi Stroke



Big Data Processing - COMP6579001

Team Members:

FERLIE HERNATA	2702231262
GIOVINCENT RICEL'S TANOTO	2702226786
JOSH NICHOLAS SUTANTO	2702234825
NIKOLAUS MARVIN LIAYASA	2702233702
RENDY RIADY	2702234421
MATTHEW ETHAN LAURENT	2702231496

BAB 1
LATAR BELAKANG

Stroke merupakan salah satu penyakit tidak menular yang terjadi akibat adanya gangguan aliran darah ke otak, baik karena penyumbatan (iskemik) maupun pecahnya pembuluh darah (hemoragik). Kondisi ini mengakibatkan terganggunya suplai oksigen dan nutrisi ke jaringan otak, sehingga sel-sel otak mulai mati dalam hitungan menit. Menurut penelitian oleh Fernandez-Lozano et al. (2024), stroke merupakan salah satu penyebab utama kematian dan kecacatan di seluruh dunia, dengan dampak signifikan terhadap kualitas hidup pasien. Dampak dari stroke bisa sangat serius, mulai dari kelumpuhan sebagian tubuh, gangguan bicara, hingga kematian. Menurut data Organisasi Kesehatan Dunia (WHO), stroke menempati peringkat kedua sebagai penyebab kematian terbanyak di dunia dan menjadi salah satu penyebab utama kecacatan jangka panjang.

Di Indonesia, prevalensi stroke terus meningkat setiap tahunnya. Berdasarkan Riset Kesehatan Dasar (Riskesdas) 2018 yang dilakukan oleh Kementerian Kesehatan RI, prevalensi stroke nasional tercatat sebesar 10,9 per mil dan menunjukkan tren kenaikan dibandingkan tahun-tahun sebelumnya. Faktor risiko utama stroke antara lain tekanan darah tinggi, diabetes, merokok, kolesterol tinggi, serta gaya hidup yang kurang aktif. Pencegahan dan deteksi dini terhadap risiko stroke menjadi langkah penting dalam menekan angka kejadian dan kematian akibat penyakit ini.

Dengan berkembangnya teknologi, khususnya dalam bidang kecerdasan buatan (Artificial Intelligence) dan big data, analisis prediktif kini dimanfaatkan untuk membantu deteksi awal penyakit seperti stroke. Dalam proses prediksi, pemilihan model klasifikasi yang tepat sangat menentukan akurasi dan keandalan hasil. Dua algoritma yang banyak digunakan dalam klasifikasi data medis adalah Random Forest dan Naive Bayes.

Random Forest merupakan algoritma *ensemble learning* yang membangun banyak pohon keputusan (decision trees) dan menggabungkan hasil prediksi dari masing-masing pohon untuk menghasilkan prediksi akhir yang lebih akurat dan stabil.Naive Bayes adalah algoritma klasifikasi berbasis probabilistik yang didasarkan pada Teorema Bayes, dengan asumsi bahwa setiap fitur (variabel input) saling independen satu sama lain terhadap label kelas. Keduanya memiliki keunggulan dan kelemahan masing-masing dalam menangani dataset dengan distribusi dan karakteristik yang berbeda.

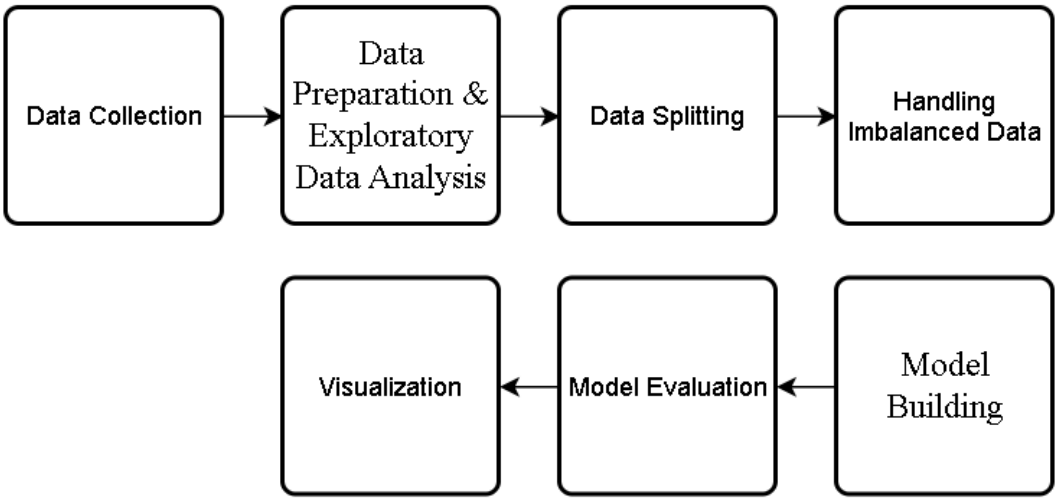
Namun, tantangan utama dalam klasifikasi data medis adalah ketidakseimbangan kelas (imbalanced dataset), di mana jumlah data pasien yang mengalami stroke jauh lebih sedikit dibandingkan dengan pasien yang sehat. Hal ini dapat menyebabkan model menjadi bias terhadap kelas mayoritas dan mengabaikan kelas minoritas yang justru lebih penting untuk diprediksi secara akurat. Oleh karena itu, diperlukan teknik khusus seperti SMOTE (Synthetic Minority Oversampling Technique) untuk menangani ketidakseimbangan ini.

Melalui penelitian ini, dilakukan perbandingan performa antara model Random Forest dan Naive Bayes dalam memprediksi kejadian stroke berdasarkan dataset yang tersedia. Penelitian ini tidak hanya bertujuan untuk menentukan model yang paling akurat, namun juga untuk mengevaluasi efektivitas teknik penyeimbangan data dalam meningkatkan kualitas prediksi.

BAB 2
METODOLOGI DAN ALUR PEKERJAAN

Dataset yang digunakan dalam penelitian ini bersumber dari Kaggle dengan judul "Stroke Prediction Dataset". Dataset ini terdiri dari beberapa atribut penting seperti usia, jenis kelamin, hipertensi, penyakit jantung, status pernikahan, pekerjaan, area tempat tinggal, dan riwayat merokok, serta variabel target berupa kejadian stroke. Variabel target ini bersifat biner, yaitu 1 untuk pasien yang mengalami stroke dan 0 untuk yang tidak.

Eksperimen dilakukan menggunakan lingkungan cloud-based Google Collab, sebuah platform yang memungkinkan eksekusi kode Python melalui browser tanpa instalasi tambahan. Pada proyek ini, runtime environment yang digunakan adalah Python 3 dengan hardware accelerator T4 GPU untuk mempercepat proses komputasi. Selain itu, Google Collab juga memungkinkan kolaborasi real-time antar anggota kelompok, sehingga seluruh anggota dapat mengakses, menjalankan, dan memodifikasi notebook secara bersama-sama selama eksperimen berlangsung. Berikut adalah Analytic Flow Big Data dalam pengerjaan proyek ini:



1. Data Collection

```
[ ]  # Menggunakan API Dataset Kaggle
     # Nama file di dalam dataset (lihat list files via CLI: `kaggle datasets files jillanisofttech/brain-stroke-dataset`)
     file_path = "brain_stroke.csv"

     # Muat ke pandas DataFrame
     df = load_dataset(
       KaggleDatasetAdapter.PANDAS,
       "jillanisofttech/brain-stroke-dataset",
       file_path
     )
```

Pada tahap ini, dataset diambil dari sumber eksternal agar dapat diproses lebih lanjut. Dalam proyek ini, dataset diperoleh dari platform Kaggle menggunakan Kaggle API melalui library kagglehub.

Prosesnya meliputi:

1. Menentukan path dataset sesuai nama file di Kaggle.
2. Menggunakan fungsi load_dataset() dari kagglehub untuk memuat file CSV ke dalam DataFrame pandas.

2. Data Preparation dan Exploratory Data Analysis

a. Data Inspection

Tahap ini kami lakukan untuk memastikan bahwa dataset yang diambil dari Kaggle sudah sesuai, tidak ada masalah struktural, dan layak untuk diproses lebih lanjut.

```
# Menampilkan data untuk mengecek apakah sudah sesuai dengan dataset kaggle atau belum
df.head()
df.shape[0] # Melihat banyaknya row data
df.info() # Mengecek tipe data tiap column
df.isnull().sum() # Mengecek apakah terdapat missing value
df.duplicated().sum() # Mengecek total duplicate
```

Berdasarkan hasil eksplorasi awal, dataset memiliki 4981 data dengan 11 variabel. Seluruh variabel tidak memiliki nilai kosong (missing value) maupun data duplikat. Tipe data terdiri dari 5 kolom kategorikal, 3 numerik, dan 3 integer. Data yang diambil juga telah sesuai dengan format dan isi data yang tercantum di Kaggle.

b. Exploratory Data Analysis (EDA)

Pada tahap ini kita menganalisis distribusi data awal, khususnya untuk variabel target stroke, guna mengidentifikasi adanya imbalance dalam dataset.

```
df['stroke' ].value_counts() # Cek total pasien yang mengalami stroke dan tidak
```

stroke	count
0	4733
1	248

dtype: int64

Hasilnya menunjukkan bahwa data bersifat imbalance, di mana jumlah pasien yang tidak mengalami stroke (0) jauh lebih banyak dibandingkan yang

mengalami stroke (1). Ketidakseimbangan ini perlu ditangani dalam tahap data preparation untuk mencegah model bias terhadap kelas mayoritas.

Tahap EDA dilanjutkan untuk memahami karakteristik data dan pola distribusi antar variabel sebelum dilakukan preprocessing dan pemodelan.

```
# ============================================================================
# VISUALISASI DATA
#
# Sub-bab:
# 1. Univariat – Variabel Kategorikal
# 2. Univariat – Variabel Kontinu (Age, BMI, Glucose)
# 3. Bivariat – Kategori vs. Stroke
# 4. Korelasi – Fitur Numerik
# 5. Imbalance – Distribusi Target (Stroke)
# ============================================================================

feature_cols_full = [
    'gender','age','hypertension','heart_disease',
    'ever_married','work_type','Residence_type',
    'avg_glucose_level','bmi','smoking_status','stroke'
]
data_df = df[feature_cols_full].copy()

num_cols = ['age','avg_glucose_level','bmi'] # numeric/kontinu fitur --> Cocok untuk countplot
cat_cols = ['gender','hypertension','heart_disease','ever_married','work_type','Residence_type','smoking_status'] # categorical feature
```

Beberapa aktivitas yang dilakukan dalam tahap ini antara lain:

1. Univariat: Visualisasi distribusi frekuensi fitur kategorikal menggunakan countplot, dan distribusi fitur numerik menggunakan histogram dan kurva KDE.

```
# ------------------------------------------------------------------------
# 1. Univariat - Variabel Kategorikal
#  Menampilkan countplot untuk tiap fitur kategorikal
# ------------------------------------------------------------------------

for col in cat_cols:
    fig, ax = plt.subplots(figsize=(12,4))            # Buat figure baru dengan ukuran 12×4 inci
    ax = sns.countplot(data=data_df, x=col)    # Gambar countplot untuk kolom saat ini
    ax.bar_label(ax.containers[0])                    # Tambahkan label angka (count) di atas setiap bar
    plt.title(f"Distribusi {col}", fontsize=14)     # Set judul plot dengan nama kolom
    plt.xlabel(col)                                          # Set label sumbu-X
    plt.ylabel("Count")                                  # Set label sumbu-Y
    plt.tight_layout()                                    # Rapikan layout agar elemen tidak terpotong
    plt.show()                                                # Tampilkan plot
```

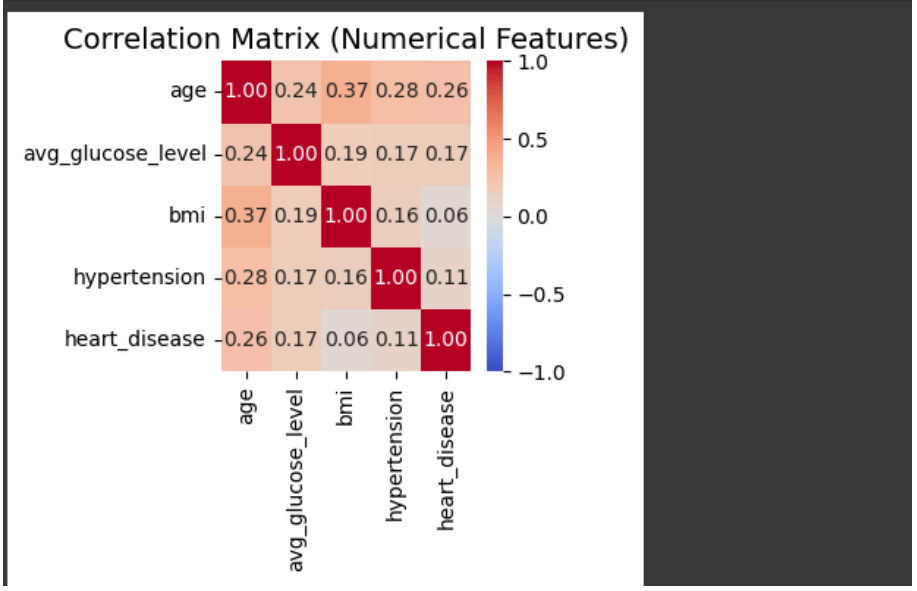
```
# ------------------------------------------------------------------------------
# 2. Univariat - Variabel Kontinu
#  Histogram + KDE untuk Age, BMI, Avg Glucose Level
# ------------------------------------------------------------------------------

for col in num_cols:
    # Buat figure & axis
    fig, ax = plt.subplots(figsize=(12,4))

    # Gambar histogram + KDE, tapi histogram-nya hitung count
    sns.histplot(
    data_df[col],      # data yang akan diplot (array-like atau pandas Series)
    kde=True,          # jika True, tambahkan kurva KDE (kernel density estimate)
    stat='count',      # hitung jumlah sampel per bin (bisa juga 'frequency', 'density', dst.)
    edgecolor='k',     # warna garis pinggir setiap bar (di sini 'k' = black)
    alpha=0.6,         # transparansi bar (0 = transparan, 1 = solid)
    ax=ax,             # axes Matplotlib tempat plot ini digambar
    )

    # Tambahkan label angka count di atas setiap bar
    ax.bar_label(ax.containers[0], fontsize=8)

    # Hanya tampilkan grid horizontal
    ax.grid(False, axis='x')

    # Judul & label
    ax.set_title(f"Distribusi {col.capitalize()}", fontsize=14)
    ax.set_xlabel(col.capitalize())
    ax.set_ylabel("Count")

    plt.tight_layout()
    plt.show()
```

2. Bivariat: Analisis hubungan antara fitur kategorikal dengan variabel target stroke menggunakan countplot dengan parameter hue, serta visualisasi hubungan fitur numerik dengan target menggunakan boxplot.

```
# ------------------------------------------------------------------------------
# 3. Bivariat - Kategori vs. Stroke
#   Countplot dengan hue='stroke' untuk melihat per-kelas
# ------------------------------------------------------------------------------

# Loop melalui setiap fitur kategorikal, tampilkan distribusi per-kelas stroke beserta label count
for col in cat_cols:
    plt.figure(figsize=(12,4))
    ax = sns.countplot(data=data_df, x=col, hue='stroke')
    [ax.bar_label(c) for c in ax.containers] # tambahkan angka count di atas tiap bar
    plt.title(f"{col.capitalize()} vs Stroke", fontsize=14) # judul plot
    plt.xlabel(col)  # label sumbu-X
    plt.ylabel("Count")   # label sumbu-Y
    plt.legend(title='Stroke', labels=['No','Yes']) # keterangan hue stroke
    plt.tight_layout()   # rapikan layout
    plt.show()

# Boxplot / Violinplot untuk hubungan continuous vs stroke
for col in num_cols:
    plt.figure(figsize=(12,4))
    sns.boxplot(data=data_df, x='stroke', y=col)
    plt.title(f"{col.capitalize()} by Stroke Status", fontsize=14) # judul plot
    plt.xlabel("Stroke")  # label sumbu-X
    plt.ylabel(col.capitalize())  # label sumbu-Y
    plt.tight_layout() # rapikan layout
    plt.show()
```

3. Korelasi: Menghitung dan memvisualisasikan matriks korelasi antar fitur numerik menggunakan heatmap.

```
# ------------------------------------------------------------------------------
# 4. Korelasi - Fitur Numerik
#   Heatmap korelasi antar fitur numerik
# ------------------------------------------------------------------------------
num_for_corr = ['age','avg_glucose_level','bmi','hypertension','heart_disease']
corr = data_df[num_for_corr].corr()

plt.figure(figsize=(4,4))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", vmin=-1, vmax=1)
plt.title("Correlation Matrix (Numerical Features)", fontsize=14)
plt.tight_layout()
plt.show()
```



4. Distribusi Target (Imbalance): Menghitung proporsi jumlah pasien yang mengalami stroke dan yang tidak, lalu divisualisasikan menggunakan pie chart.

```
# ------------------------------------------------------------------------------
# 5. Imbalance - Distribusi Target (Stroke)
#   Pie chart untuk menunjukkan proporsi kelas
# ------------------------------------------------------------------------------
counts = data_df['stroke'].value_counts()
labels = ['No Stroke','Stroke']

plt.figure(figsize=(3,3))
plt.pie(counts, labels=labels, autopct="%1.1f%%", startangle=90, explode=(0,0.1))
plt.title("Proporsi Pasien dengan/ tanpa Stroke", fontsize=14)
plt.tight_layout()
plt.show()
```

Hasil EDA menunjukkan adanya ketidakseimbangan data pada variabel target, serta beberapa fitur yang memiliki distribusi tidak merata, yang kemudian menjadi pertimbangan dalam proses data preparation dan pemodelan.

c. Data Preparation untuk Modeling

```
feature_cols = [
    'gender','age','hypertension','heart_disease',
    'ever_married','work_type','Residence_type',
    'avg_glucose_level','bmi','smoking_status'
]
X = df[feature_cols]    # hanya fitur
y = df['stroke']        # label/target

# Handling missing values
# Best practies buat jaga jaga

numeric_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),  # isi median jika ada NaN
    ('scaler',  StandardScaler()) # ubah dari mean ke var, mean = 0, std = 1
])

# Preprocessor ==> Menggabungkan langkah preprocessing numerik & kategorikal
preprocessor = ColumnTransformer([
    ('num', numeric_pipeline, num_cols), # Numerik, pipeline: imputasi median + standard scaler
    ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols) # Categorical, ubah kategorikal menjadi numerik
])
```

Pada tahap ini, dilakukan beberapa proses untuk mempersiapkan dataset sebelum digunakan dalam pemodelan:

1. Memisahkan Fitur dan Target

   Variabel target stroke dipisahkan dari fitur-fitur prediktor lainnya.

2. Handling Missing Value dan Normalisasi Data

   Untuk fitur numerik, diterapkan:

- SimpleImputer untuk mengisi missing value dengan nilai median.
- StandardScaler untuk melakukan standarisasi data numerik agar memiliki mean 0 dan variansi 1.

3. Encoding Data Kategorikal

   Menggunakan OneHotEncoder untuk mengubah data kategorikal menjadi numerik, dengan opsi handle_unknown='ignore' untuk menangani kategori baru saat testing.

4. ColumnTransformer

   Menyatukan pipeline numerik dan encoder kategorikal dalam satu preprocessor:

3. Data Splitting

   Setelah dilakukan proses data preparation, dataset dibagi menjadi data latih dan data uji. Proses ini dilakukan menggunakan fungsi train_test_split dari library scikit-learn dengan rasio 80:20, di mana 80% data digunakan untuk melatih model dan 20% sisanya digunakan untuk menguji performa model. Pemisahan data ini bertujuan untuk memastikan bahwa model dievaluasi menggunakan data yang tidak pernah dilihat sebelumnya, sehingga hasil evaluasi dapat merefleksikan performa model secara objektif.

```
#Memisahkan data X dan y asli menjadi set pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True)

# Terapkan praprosesor ke X_train dan X_test
# Pasangkan preprosesor hanya pada data pelatihan (X_train) untuk menghindari kebocoran data
X_train_processed = preprocessor.fit_transform(X_train)
# Transformasikan data uji menggunakan preprosesor yang dipasang pada data pelatihan
X_test_processed = preprocessor.transform(X_test)
```

   Selanjutnya data latih diproses menggunakan fit_transform, sedangkan data uji hanya menggunakan transform untuk menghindari kebocoran data (data leakage).

4. Handling Imbalanced Data

```
#Memisahkan data X dan y asli menjadi set pelatihan dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True)

# Terapkan praprosesor ke X_train dan X_test
# Pasangkan preprosesor hanya pada data pelatihan (X_train) untuk menghindari kebocoran data
X_train_processed = preprocessor.fit_transform(X_train)
# Transformasikan data uji menggunakan preprosesor yang dipasang pada data pelatihan
X_test_processed = preprocessor.transform(X_test)

# Oversampling dengan SMOTE (hanya di data train)
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_processed, y_train)

# Cek distribusi setelah SMOTE (optional)
from collections import Counter
print("Distribusi setelah SMOTE:", Counter(y_train_smote))
```

Untuk mengatasi masalah data yang tidak seimbang (imbalanced), kita menerapkan metode Synthetic Minority Oversampling Technique (SMOTE) pada data latih. SMOTE bekerja dengan cara membuat data sintetis dari kelas minoritas berdasarkan nilai fitur dari data yang ada. Proses oversampling hanya dilakukan pada data latih agar model dapat belajar dari distribusi data yang seimbang tanpa memengaruhi data uji. Setelah proses SMOTE, distribusi kelas pada data latih dicek kembali untuk memastikan bahwa distribusinya telah seimbang.

5. Model Building

```
# Buat model Random Forest dengan 20 pohon dan kriteria entropy
rand_clf = RandomForestClassifier(n_estimators=20, criterion="entropy", random_state=42)
# Latih model pakai data training yang sudah diproses
rand_clf.fit(X_train_smote, y_train_smote)

# Evaluasi di data uji (tanpa SMOTE)
# SMOTE hanya diterapkan pada data pelatihan agar model belajar dari distribusi seimbang, sedangkan evaluasi tetap
# menggunakan data uji asli tanpa SMOTE agar metrik mencerminkan performa nyata.
y_pred = rand_clf.predict(X_test_processed)

# Buat model Naive Bayes
NaiveBayes = GaussianNB()
# Latih model pakai data training yang sudah diproses (udah balance)
NaiveBayes.fit(X_train_smote, y_train_smote)

# Prediksi label untuk data uji yang sudah diproses
y_pred = NaiveBayes.predict(X_test_processed)
```

Tahap modeling dilakukan dengan membangun dua algoritma klasifikasi yaitu Random Forest Classifier dan Naive Bayes Classifier. Kedua model dilatih menggunakan data latih yang telah diproses dan diseimbangkan dengan metode SMOTE.

Model Random Forest dibangun menggunakan 20 pohon keputusan (n_estimators=20) dan kriteria pemilihan split berupa entropy. Model ini dilatih dengan data latih hasil oversampling untuk mengurangi bias terhadap kelas mayoritas. Setelah itu, dilakukan inisialisasi model Naive Bayes menggunakan pendekatan

distribusi Gaussian. Model ini juga dilatih dengan data latih hasil SMOTE untuk menjaga konsistensi distribusi kelas saat pelatihan.

6. Model Evaluation

   Tahap akhir evaluasi dilakukan dengan membandingkan performa kedua model menggunakan beberapa metrik evaluasi, yaitu Accuracy, Precision, Recall, dan F1-Score. Nilai metrik dihitung berdasarkan hasil prediksi terhadap data uji.

   Langkah pertama adalah melakukan prediksi menggunakan model Random Forest dan Naive Bayes. Hasil prediksi kemudian dibandingkan dengan label asli untuk menghitung nilai masing-masing metrik. Setelah itu, dilakukan perhitungan metrik evaluasi untuk kedua model dengan menggunakan fungsi accuracy_score, precision_score, recall_score, dan f1_score dari library scikit-learn. Hasil evaluasi disusun dalam bentuk tabel perbandingan menggunakan pandas.DataFrame untuk memudahkan analisis performa kedua model.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import pandas as pd
import matplotlib.pyplot as plt

# --- 1. Dapatkan prediksi kedua model ---
y_pred_rf = rand_clf.predict(X_test_processed)
y_pred_nb = NaiveBayes.predict(X_test_processed)

# --- 2. Hitung metrik untuk masing-masing model ---
metrics = []
for name, y_pred in [("Random Forest", y_pred_rf),
                     ("Naive Bayes",    y_pred_nb)]:
    metrics.append({
        'Model':     name,
        'Accuracy':  accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred, average='macro', zero_division=0),
        'Recall':    recall_score(y_test, y_pred, average='macro', zero_division=0),
        'F1 Score':  f1_score(y_test, y_pred, average='macro', zero_division=0)
    })

# --- 3. Buat DataFrame dan tampilkan ---
df_metrics = pd.DataFrame(metrics)
print(df_metrics.to_string(index=False))
```

7. Visualization

```python
import matplotlib.pyplot as plt

models = df_metrics['Model'].tolist()
scores = df_metrics[['Accuracy','Precision','Recall','F1 Score']]

x = list(range(len(models)))
width = 0.2

# Buat figure & axis
fig, ax = plt.subplots(figsize=(8, 5))

# Simpan objek bar untuk setiap metrik
bar_containers = []
for i, metric_name in enumerate(scores.columns):
    bars = ax.bar(
        [xi + i*width for xi in x],
        scores[metric_name],
        width=width,
        label=metric_name
    )
    bar_containers.append(bars)

# Tambahkan label nilai di atas bar (format 2 desimal)
for bars in bar_containers:
    ax.bar_label(bars, fmt='%.2f', padding=3)

# Atur label, judul, dan legenda
ax.set_xlabel('Model')
ax.set_ylabel('Score')
ax.set_title('Perbandingan Kinerja Model')
ax.set_xticks([xi + 1.5*width for xi in x])
ax.set_xticklabels(models)
ax.legend()

plt.tight_layout()
plt.show()
```

Untuk mempermudah interpretasi hasil evaluasi model, dilakukan visualisasi perbandingan nilai Accuracy, Precision, Recall, dan F1-Score menggunakan grafik batang. Visualisasi ini menampilkan performa masing-masing model dalam satu plot, sehingga perbedaan kinerja antar model dapat terlihat dengan jelas.

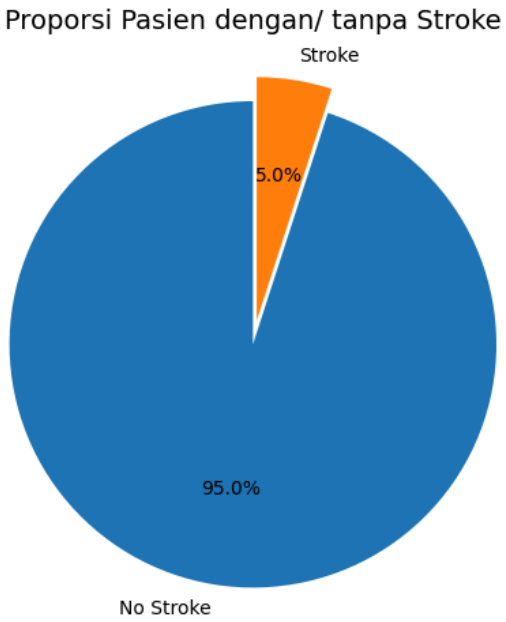
Grafik dibuat menggunakan library matplotlib, dengan setiap metrik ditampilkan dalam bar yang berbeda warna untuk tiap model. Nilai skor ditampilkan di atas masing-masing batang untuk memudahkan pembacaan hasil. Visualisasi ini berfungsi sebagai alat bantu analisis untuk melihat model mana yang memiliki performa lebih baik berdasarkan berbagai metrik evaluasi.

BAB 3
EVALUASI DAN DETAIL DARI ALUR PEKERJAAN

Setelah membangun dua model klasifikasi, yaitu Random Forest dan Naive Bayes, evaluasi dilakukan untuk menilai performa kedua model dalam memprediksi kejadian stroke. Evaluasi ini dilakukan menggunakan data uji dengan berbagai metrik seperti Accuracy, Precision, Recall, dan F1 Score. Selain itu, hasil evaluasi juga dianalisis menggunakan beberapa visualisasi untuk memperkuat interpretasi terhadap kinerja model.

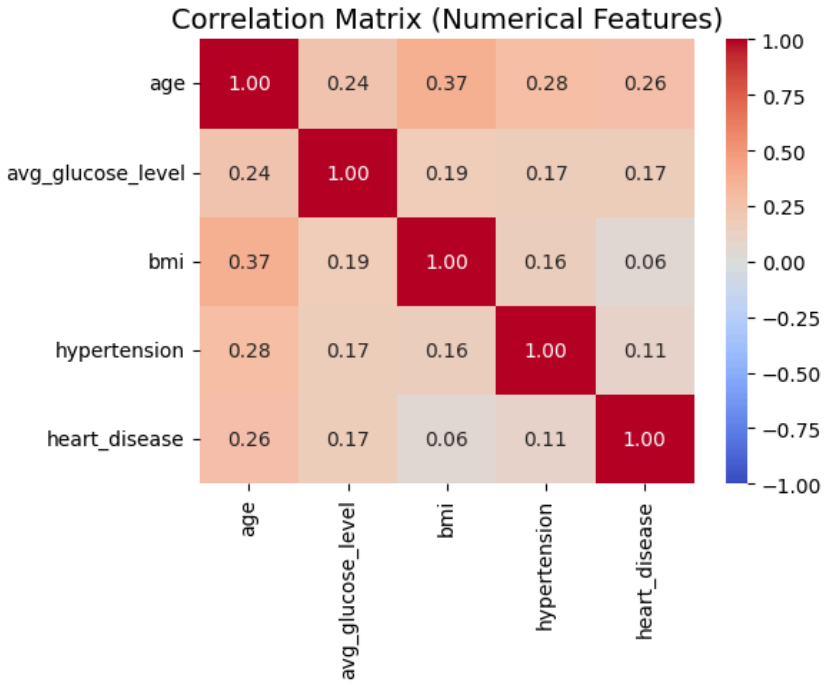
1. Distribusi Kelas pada Data Asli

   Distribusi target pada dataset menunjukkan bahwa data sangat tidak seimbang. Visualisasi berupa pie chart memperlihatkan bahwa mayoritas data merupakan pasien yang tidak mengalami stroke (label 0).



   Grafik menunjukkan bahwa jumlah pasien yang tidak mengalami stroke jauh lebih besar dari yang mengalami stroke, dengan rasio mendekati 19:1.
   Ketidakseimbangan ini sangat mempengaruhi performa model klasifikasi, karena algoritma cenderung memprioritaskan prediksi terhadap kelas mayoritas untuk mencapai akurasi tinggi, namun mengabaikan kelas minoritas yang justru lebih penting (stroke = 1).

2. Korelasi Antar Fitur (Heatmap)



Sebelum proses pemodelan, dilakukan analisis korelasi antar fitur numerik untuk mengetahui hubungan antar variabel. Visualisasi berupa heatmap menunjukkan bahwa:

- Fitur age dan avg_glucose_level memiliki korelasi positif terhadap stroke.
- Fitur bmi dan variabel lainnya menunjukkan korelasi lemah atau hampir tidak ada.

Hasil ini mengindikasikan bahwa tidak ada fitur tunggal yang dominan dalam memprediksi stroke, sehingga pendekatan model ensemble seperti Random Forest lebih cocok dibanding model yang mengandalkan independensi fitur seperti Naive Bayes.

3. Penanganan Ketidakseimbangan Data (SMOTE)

```
# Cek distribusi setelah SMOTE (optional)
from collections import Counter
print("Distribusi setelah SMOTE:", Counter(y_train_smote))
```

```
Distribusi setelah SMOTE: Counter({0: 3790, 1: 3790})
```

Untuk mengatasi masalah ketidakseimbangan data (imbalanced dataset), proyek ini

menerapkan teknik SMOTE (Synthetic Minority Oversampling Technique) pada data latih. SMOTE bekerja dengan membuat data sintetis untuk kelas minoritas (stroke = 1) berdasarkan karakteristik data yang sudah ada.

Langkah ini dilakukan hanya pada data latih agar tidak menyebabkan data leakage ke data uji. Tujuan utama penerapan SMOTE adalah untuk membantu model belajar dari distribusi kelas yang lebih seimbang, sehingga model tidak terlalu bias terhadap kelas mayoritas (non-stroke).

Tanpa penanganan imbalance, model cenderung mengabaikan kelas minoritas dan menghasilkan nilai recall yang sangat rendah, karena model lebih fokus pada prediksi kelas mayoritas yang dominan.

Penerapan SMOTE dalam pipeline preprocessing telah membantu meningkatkan sensitivitas model terhadap kelas minoritas, walaupun masih ada tantangan dalam mencapai nilai F1-score yang optimal.

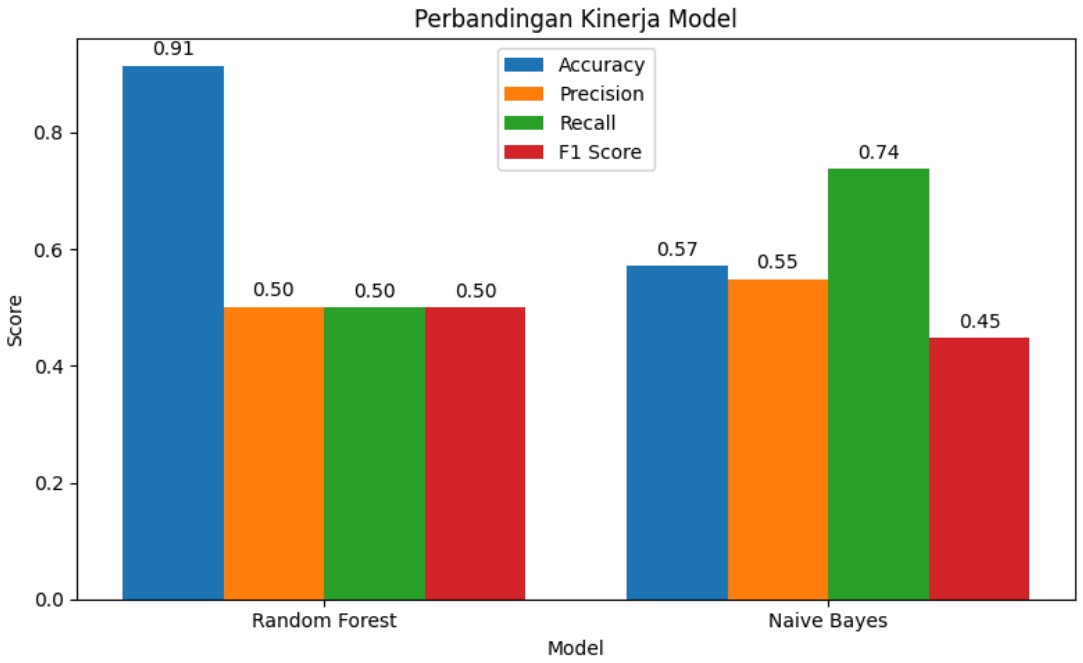
4. Evaluasi Kinerja Model

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	0.9147	0.5015	0.5010	0.5002
Naive Bayes	0.5707	0.5489	0.7381	0.4487

Berdasarkan tabel di atas, Random Forest menunjukkan nilai accuracy tertinggi (91.47%) namun memiliki recall dan F1-score yang rendah, menandakan bahwa model ini lebih sering benar dalam memprediksi kelas mayoritas (non-stroke) namun kurang mampu mengenali kasus stroke (minoritas). Sementara itu, Naive Bayes memiliki recall tertinggi (73.81%) yang berarti model lebih “sensitif” terhadap kasus stroke, namun akurasi keseluruhannya rendah, menunjukkan banyak prediksi salah di kelas lain.

5. Visualisasi Evaluasi Model

Untuk memudahkan perbandingan, hasil evaluasi divisualisasikan dalam bentuk grafik batang berikut:



Visualisasi metrik evaluasi dalam bentuk grafik batang menunjukkan perbedaan performa yang signifikan antara kedua model. Grafik ini menampilkan empat metrik utama:

- Accuracy lebih tinggi pada Random Forest.
- Recall tertinggi ada di Naive Bayes.
- Precision dan F1-score relatif rendah pada kedua model, yang menandakan bahwa prediksi stroke masih banyak kesalahan baik dalam bentuk false positives maupun false negatives.

Grafik batang ini sangat penting karena memperlihatkan bahwa Random Forest unggul secara keseluruhan, namun Naive Bayes lebih baik dalam deteksi positif stroke, yang bisa menjadi pertimbangan tergantung kebutuhan (preventif atau general).

6. Interpretasi Akhir
    - Random Forest adalah model yang secara umum lebih stabil, namun kurang sensitif terhadap kelas minoritas.
    - Naive Bayes memiliki performa recall yang baik, namun menghasilkan banyak kesalahan klasifikasi.
    - Penerapan SMOTE efektif menyeimbangkan data latih, namun tidak sepenuhnya mengatasi tantangan klasifikasi karena distribusi asli data uji tetap tidak seimbang.

- Perlu pertimbangan lebih lanjut untuk meningkatkan F1-score, seperti:
    - Feature engineering tambahan
    - Algoritma lain seperti XGBoost atau LightGBM
    - Penggunaan threshold tuning atau cost-sensitive learning

BAB 4
KESIMPULAN DAN SARAN

1. Kesimpulan

   Berdasarkan hasil eksperimen dan evaluasi yang telah dilakukan dalam proyek ini, dapat disimpulkan beberapa poin penting sebagai berikut:

   - Model Random Forest menunjukkan performa terbaik dari segi akurasi, yaitu sebesar 91.47%, namun memiliki kelemahan pada recall dan F1-score yang rendah. Hal ini menandakan bahwa model lebih cenderung memprediksi kelas mayoritas (non-stroke) dengan benar, tetapi kurang mampu mendeteksi kasus stroke (kelas minoritas).
   - Model Naive Bayes memiliki performa recall yang lebih tinggi dibandingkan Random Forest, yaitu sebesar 73.81%, namun mengalami penurunan signifikan pada akurasi dan precision, yang menunjukkan banyaknya prediksi false positive.
   - Ketidakseimbangan kelas pada dataset stroke menjadi tantangan utama dalam proses klasifikasi. Untuk mengatasi hal ini, diterapkan metode SMOTE (Synthetic Minority Oversampling Technique) pada data latih, yang berhasil menyeimbangkan distribusi kelas selama proses pelatihan dan membantu meningkatkan performa model, terutama dalam mendeteksi kelas minoritas.
   - Visualisasi korelasi antar fitur menunjukkan bahwa tidak ada fitur tunggal yang dominan dalam mempengaruhi prediksi stroke. Oleh karena itu, penggunaan model yang dapat menangani banyak fitur dan interaksi kompleks seperti Random Forest menjadi lebih relevan.
   - Secara keseluruhan, meskipun performa model masih belum sempurna, kedua algoritma memiliki potensi untuk digunakan dalam sistem pendukung keputusan medis, dengan catatan dilakukan penyesuaian lebih lanjut.

2. Saran

   Beberapa saran yang dapat diberikan untuk pengembangan dan penelitian lebih lanjut antara lain:

- Penggunaan Algoritma Tambahan: Mencoba model lain seperti XGBoost, LightGBM, atau SVM yang sering menunjukkan performa lebih baik dalam klasifikasi pada dataset yang kompleks dan tidak seimbang.
- Tuning Hyperparameter: Melakukan pencarian kombinasi hyperparameter optimal pada setiap model untuk meningkatkan kinerja, terutama untuk mengurangi overfitting dan underfitting.
- Feature Engineering: Menambahkan fitur-fitur baru atau melakukan transformasi terhadap fitur yang ada agar model lebih mudah menemukan pola dalam data.
- Threshold Tuning: Menyesuaikan ambang batas klasifikasi untuk memperbaiki trade-off antara precision dan recall, terutama jika tujuan aplikasi lebih sensitif terhadap kesalahan tertentu (misalnya, mendeteksi semua kasus stroke walaupun menghasilkan false positives).
- Validasi K-Fold: Menggunakan teknik validasi silang (cross-validation) agar hasil evaluasi lebih stabil dan tidak bergantung pada satu pemisahan data uji.
- Penerapan pada Data Nyata: Jika memungkinkan, model perlu diuji pada data medis riil yang lebih besar dan beragam untuk memastikan kelayakan implementasi dalam praktik klinis.

Referensi

https://doi.org/10.48550/arXiv.2402.00638

https://arteri.sinergis.org/arteri/article/download/